PWM Design Brief

Our pwm_timer module is a multi-channel, programmable PWM and timer generator with Wishbone bus support. It supports 4 independent PWM outputs with individually configurable duty cycles and periods, controlled either via internal registers or an external duty cycle input.

Key Features:

- **4 PWM Channels** (o_pwm[3:0]) with:
  - Separate Period and Duty Cycle registers
  - Optional external DC input (i_DC)
- **Dual Clock Source:**
  - Select between system clock (i_clk) and external clock (i_extclk) via control bit
- **Clock Divider:**
  - Adjustable clock rate using Divisor
  - Handling Invalid Divisor Values
- **Dual Operation Modes:**
  - PWM Mode: Generates pulse-width-modulated signals
  - Timer Mode: Acts like a timer with interruption including the ability to determine which channel caused interruption and one-shot/continuous run
- **Wishbone Interface:**
  - Supports register reads/writes via i_wb_* signals
- **Safe DC Handling:**
  - Clips Duty Cycle to not exceed Period (safe_DC)

Tricky Challenges:

1- **Challenge:** Managing multiple PWM outputs with distinct Period/DC values while sharing the clock and control logic.
Solution: We used a generate block to iterate over the 4 channels (genvar ch), creating per-channel logic with shared clocking. Each channel uses: Its own main counter[ch], DC [ch], Period[ch] Shared Ctrl settings and clock divider This avoids code duplication and ensures consistent timing across all channels while maintaining configurability.

2- **Challenge:** In PWM and timer systems, invalid settings — such as a zero divisor, or a duty cycle greater than the period — can lead to unstable behavior, glitches, or even total failure of signal generation.
Solution: We implemented built-in safety mechanisms to detect and correct these issues at runtime Duty cycles are automatically clamped to the value of the period to prevent incorrect PWM output

3- **Challenge:** While building the testbench, we realized that manually writing Wishbone transactions and control sequences for each scenario (e.g., setting period, duty cycle, switching modes, testing interrupts or down-clocking) would lead to *repetitive and lengthy code*, making the testbench harder to manage and extend.

Solution: We addressed this by creating reusable tasks and functions These abstractions allowed us to replace large code blocks with simple, modular calls, improving readability and scalability. This structure made it easier to add or modify test cases without cluttering the codebase or duplicating logic.

Test Sequence:

The PWM timer testbench validates a PWM/Timer module with Wishbone bus interface. It runs 8 comprehensive tests covering PWM generation, timer interrupts, clock division, and error handling.

Test 1: Wishbone Operations : Validates basic bus read/write functionality by configuring period to 8 and duty cycle to 6, then enabling PWM mode and counter. Ensures the Wishbone interface works correctly for register access.

Test 2: PWM 50% Duty Cycle : Tests standard PWM generation with period of 100 cycles and 50% duty cycle. Enables full PWM output and verifies the signal alternates between high and low states for equal durations.

Test 3: PWM 25% Duty Cycle : Demonstrates real-time duty cycle modification by changing from 50% to 25% while PWM is running. Shows the system can dynamically adjust PWM parameters without stopping operation.

Test 4: Timer Interrupt : Switches to timer mode with a short period of 50 cycles to test interrupt generation. Verifies that the interrupt flag gets set when the timer reaches its period value and displays the result.

Test 5: Timer Clear : Tests interrupt flag clearing mechanism by continuing timer operation and confirming the interrupt flag automatically clears for subsequent timer cycles.

Test 6: Clock Divisor /2 : Tests clock division functionality by setting divisor to 2, which makes the PWM run at half speed. Uses period of 20 and duty cycle of 10 to maintain 50% duty cycle at the divided clock rate.

Test 7: Clock Divisor /10 : Further tests clock division with divisor of 10, making PWM run at 1/10 normal speed. Uses period of 10 and duty cycle of 5 to verify proper operation at heavily divided clock rates.

Test 8: Edge Cases : Comprehensive boundary condition testing with three sub-tests:

- Tests duty cycle greater than period (expects PWM to stay constantly high)
- Tests divisor of zero (expects system to operate as if divisor is 1 for protection)
- Tests manual counter reset functionality during operation

GitHub Project Repo Link : https://github.com/MohGendy/PWM-Timer